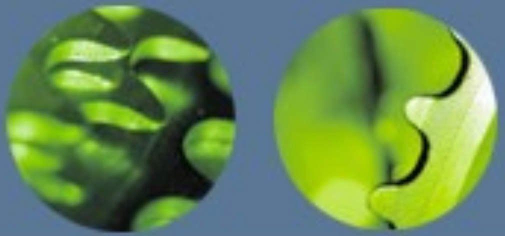
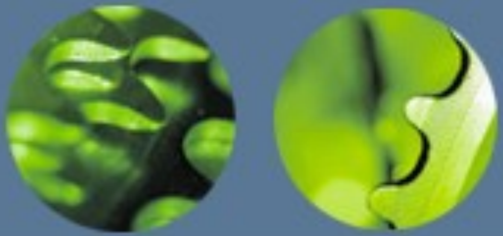


Informed Search (Heuristic Search)

- ✓ In uninformed search, we don't try to evaluate which of the nodes on the frontier are most promising.
- ✓ We never "look-ahead" to the goal.
- ✓ Informed search have problem specific knowledge apart from problem definition.
- ✓ Use of Heuristic improves efficiency of search process.
- ✓ The idea is to develop a domain specific heuristic function h(n).
- ✓ h(n) guesses the cost of getting to the goal from node n.

Informed Search (Heuristic Search)

- ✓ Greedy Best First Search
- ✓ A* Search
- ✓ Hill Climbing Search
- ✓ Simulated Annealing Search



Best First Search

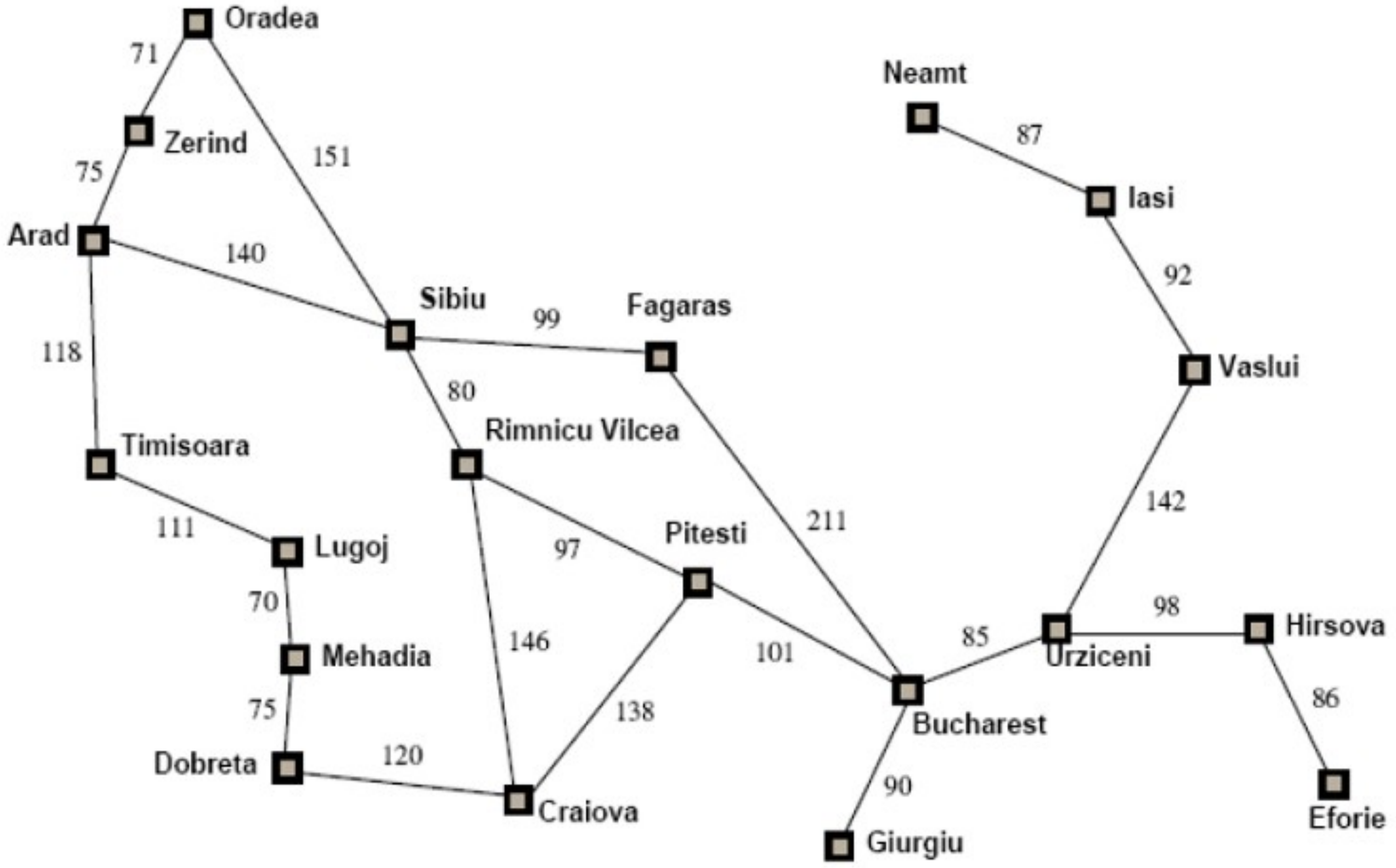
- ✓ A node is selected for expansion based on evaluation function f(n)
- ✓ Node with lowest evaluation function is expanded first.
- ✓ The evaluation function must represent some estimate of the cost of the path from state to the closest goal state.
- ✓ One of the important heuristic function is h(n)
- ✓ h(n) is the estimate cost of the cheapest path from node to the goal
- ✓ Types
  - Greedy Best First Search
  - A* Search

Greedy Best-First Search

- ✓ Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly.
- ✓ Thus, it evaluates nodes by using just the heuristic function; that is, $f(n) = h(n)$.

Example

Task: Find a path from Arad to Bucharest

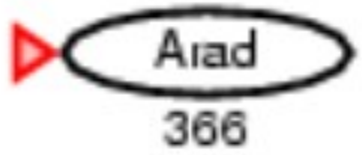


Straight–line distance to Bucharest

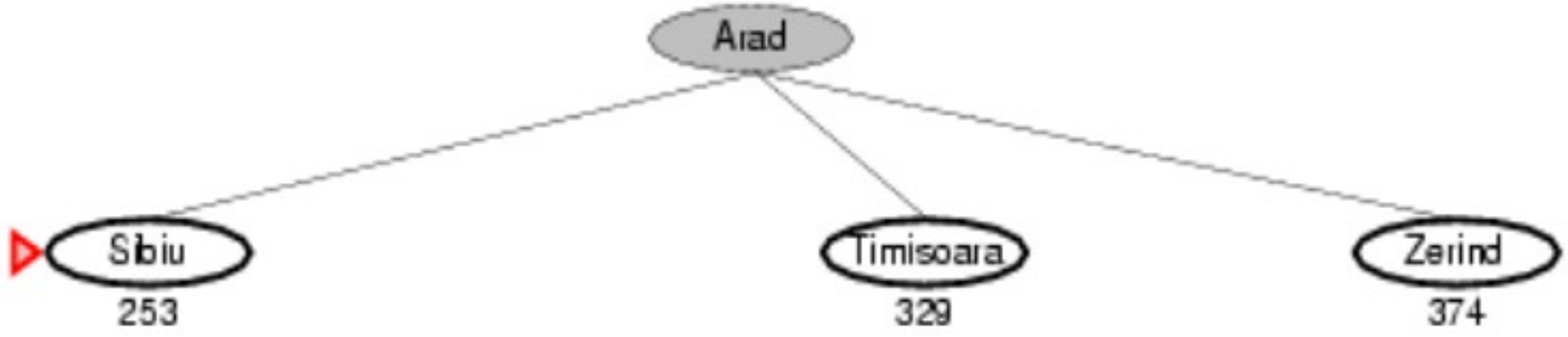
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Example

- ✓ Start from source node

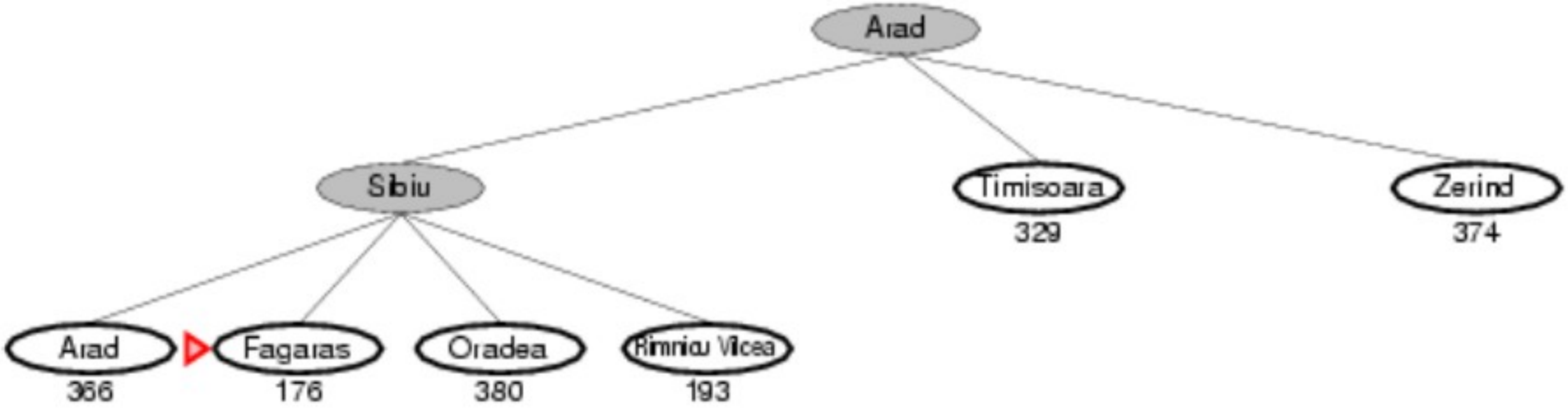


- ✓ Evaluate cities connected to source and choose the one with lowest h(distance) value.



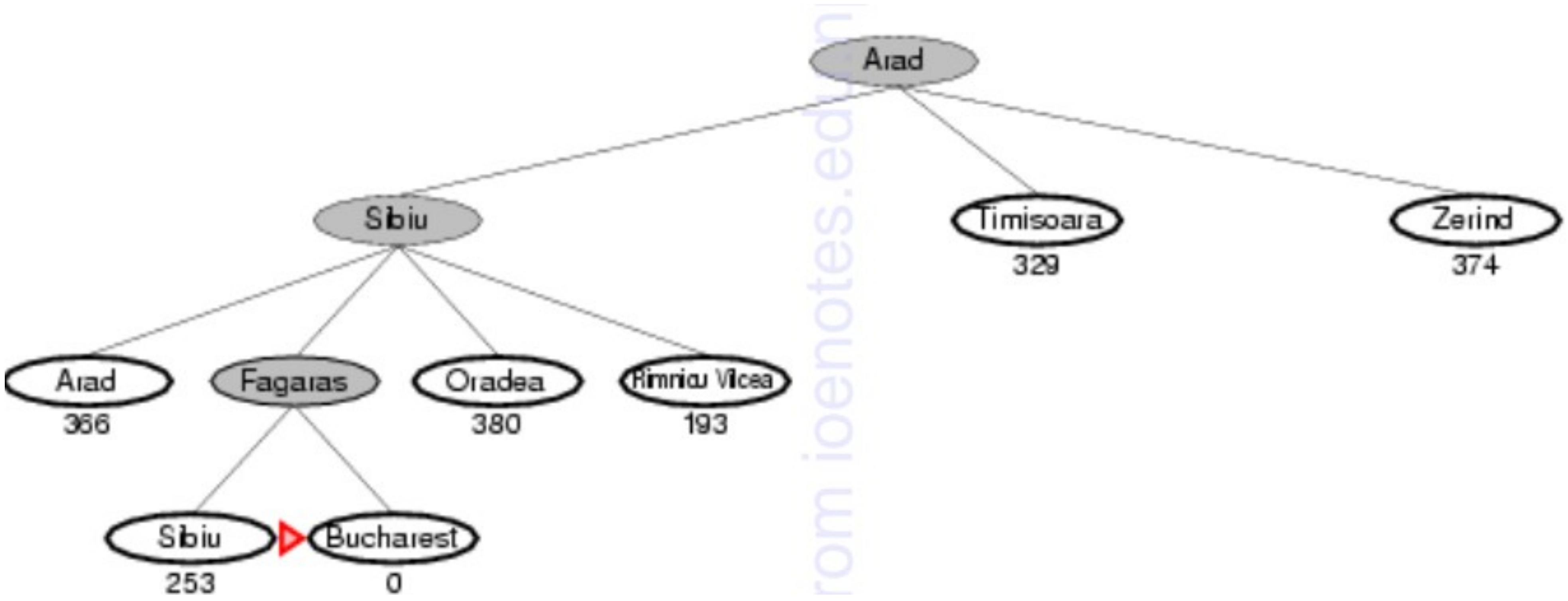
Example

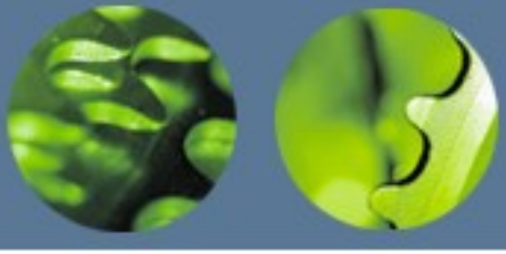
- ✓ Continue evaluating cities with their distance to destination



Example

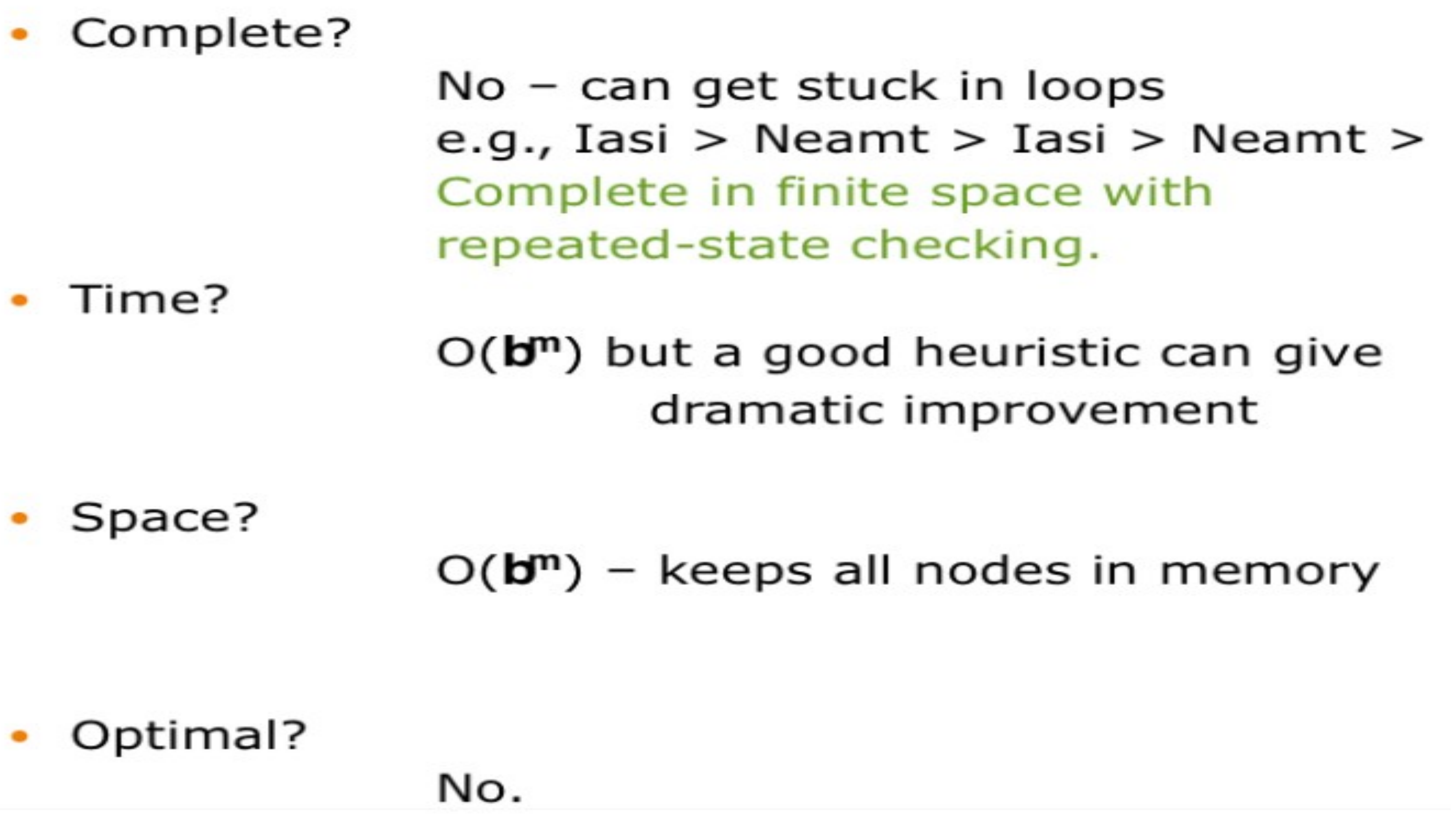
- ✓ Continue evaluating until destination is reached





Greedy Best-First Search

Properties of Greedy Best-First Search



- Complete?

  No – can get stuck in loops
  e.g., Iasi > Neamt > Iasi > Neamt >
  Complete in finite space with repeated-state checking.

- Time?

  O(b^m) but a good heuristic can give dramatic improvement

- Space?

  O(b^m) – keeps all nodes in memory

- Optimal?

  No.

Greedy Best-First Search

- Advantages:

✔Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.

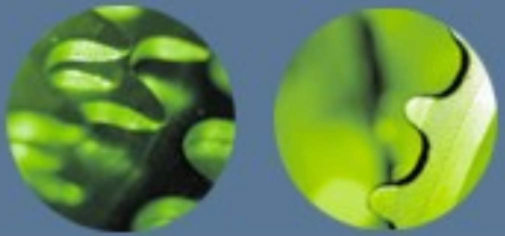
✔This algorithm is more efficient than BFS and DFS algorithms.

- Disadvantages:

✔It can behave as an unguided depth-first search in the worst case scenario.

✔It can get stuck in a loop.

✔This algorithm is not optimal.



A* Search

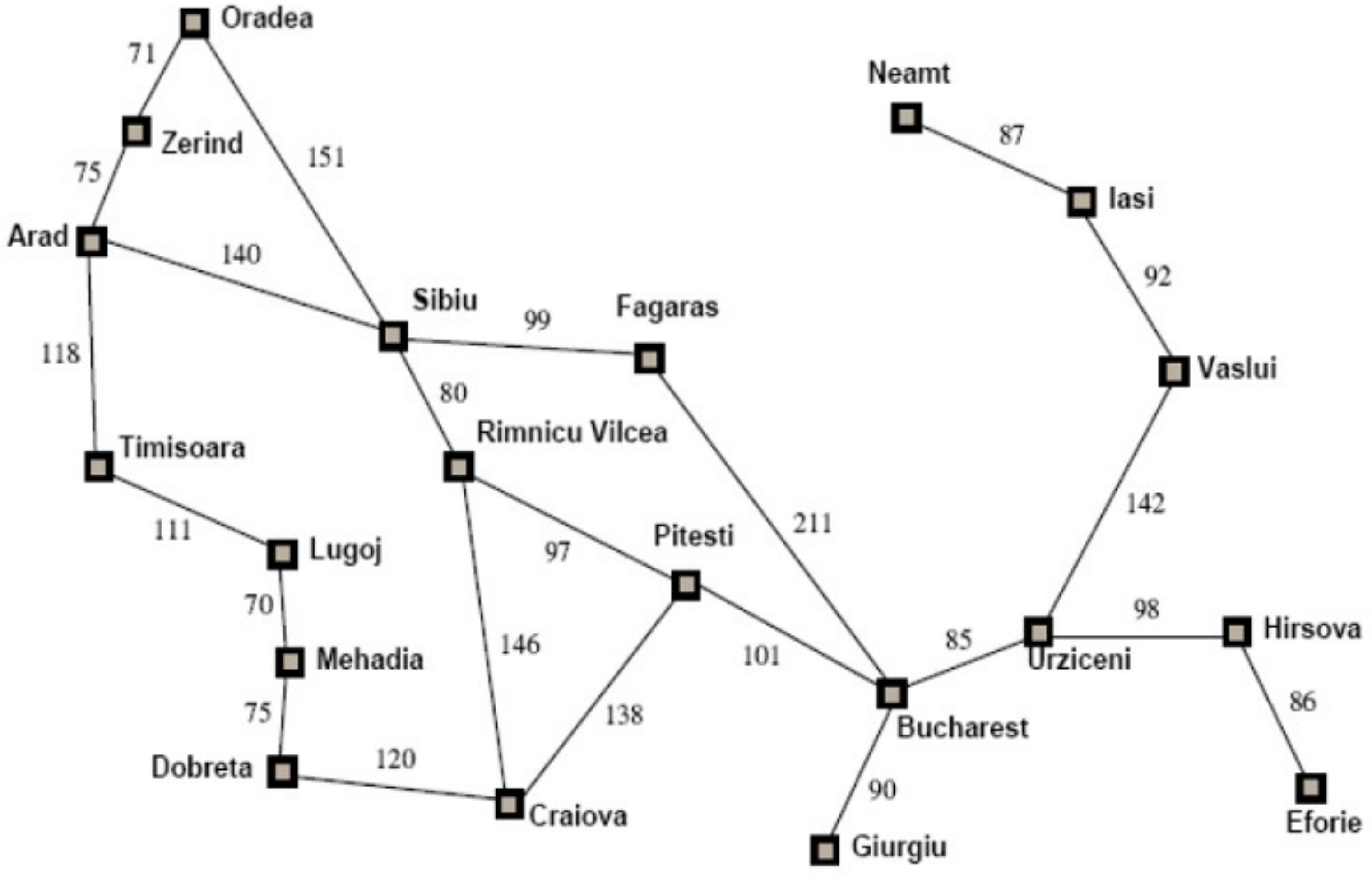
- ✓ Greedy best first search analyses nodes with lowest cost of node n to reach goal.
- ✓ And it may get stuck in search of goal.
- ✓ Idea : avoid expanding paths that are already expensive

Evaluation function f(n) = g(n) + h(n)

- g(n) = cost so far to reach n
- h(n) = estimated cost from n to goal
- f(n) = estimated total cost of path through n to goal

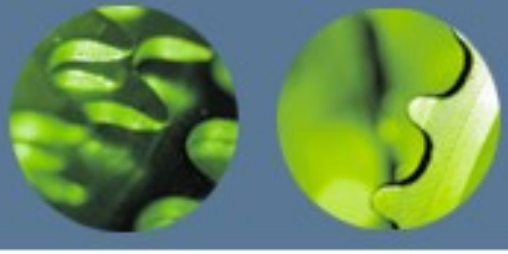
Example

Task: Find a path from Arad to Bucharest



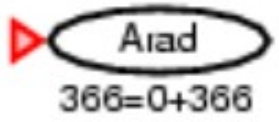
Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

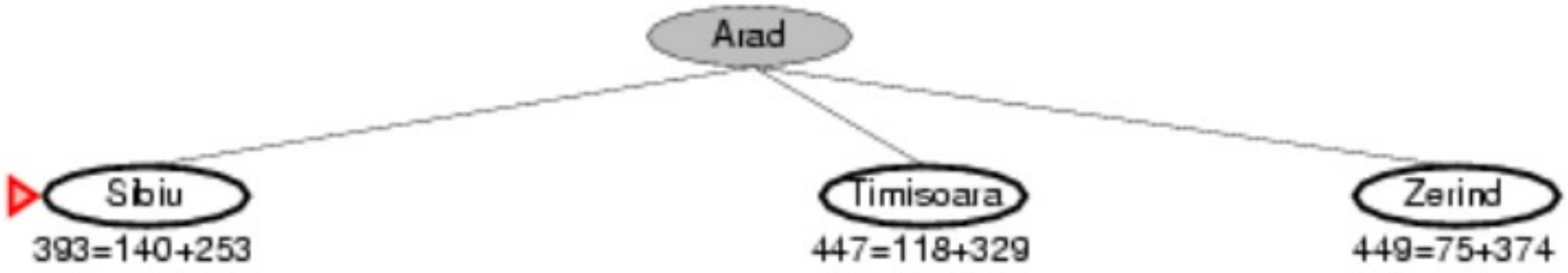


Example

- ✓ Start with source node.

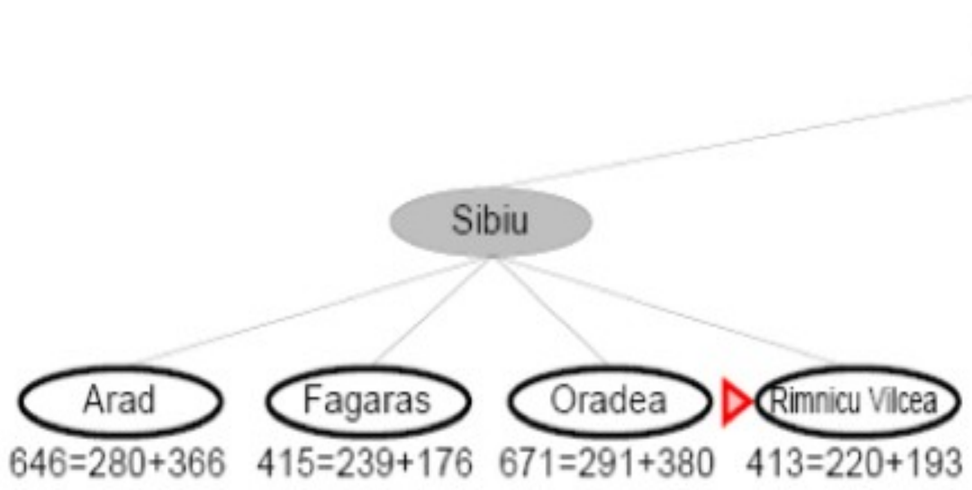


- ✓ Evaluate nodes connected to source. Evaluate f(n)=g(n)+h(n) for each node. Select node with lowest f(n) value. Sibiu node is chosen

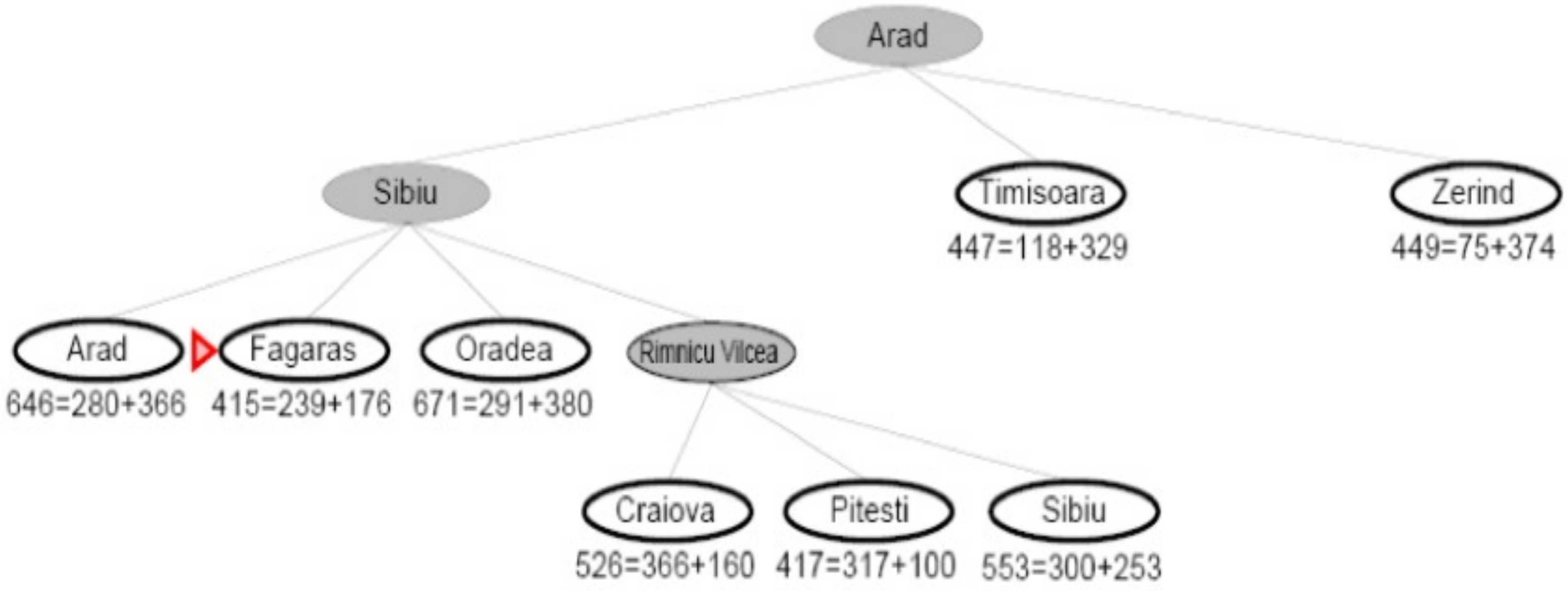


Example

✔ Rimniou vicea node has lowest f(n) value, so this city is chosen

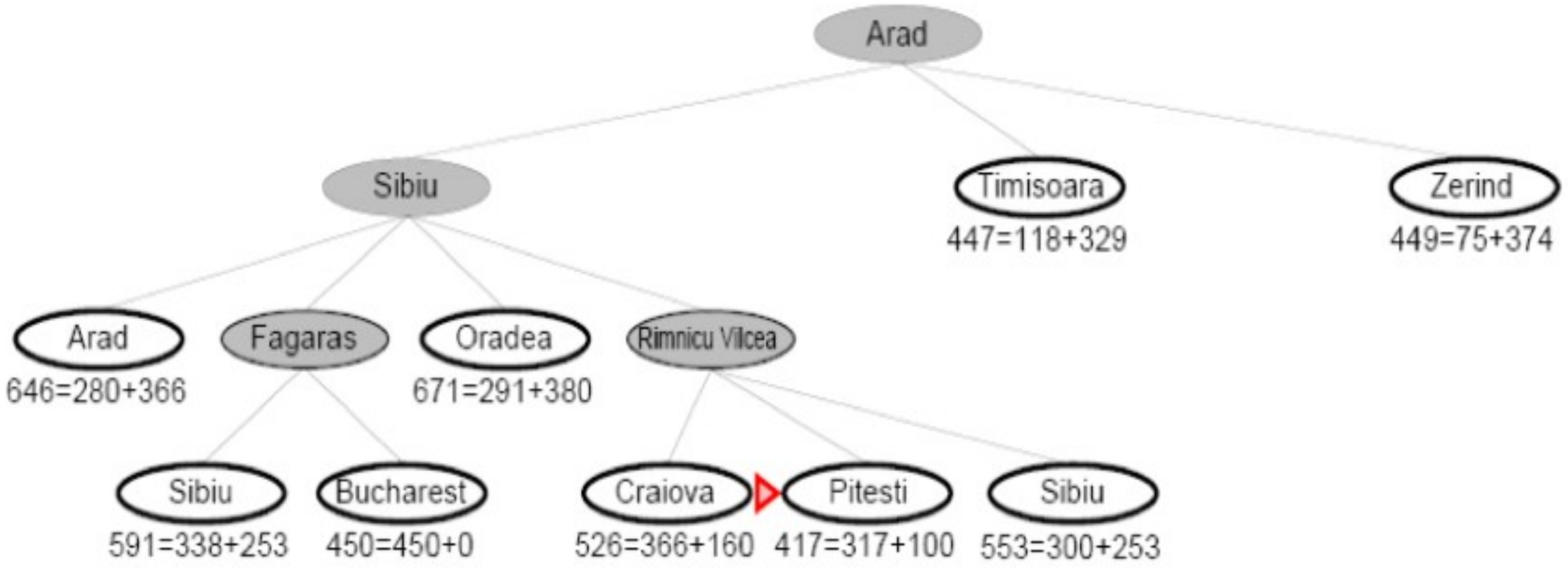


Example



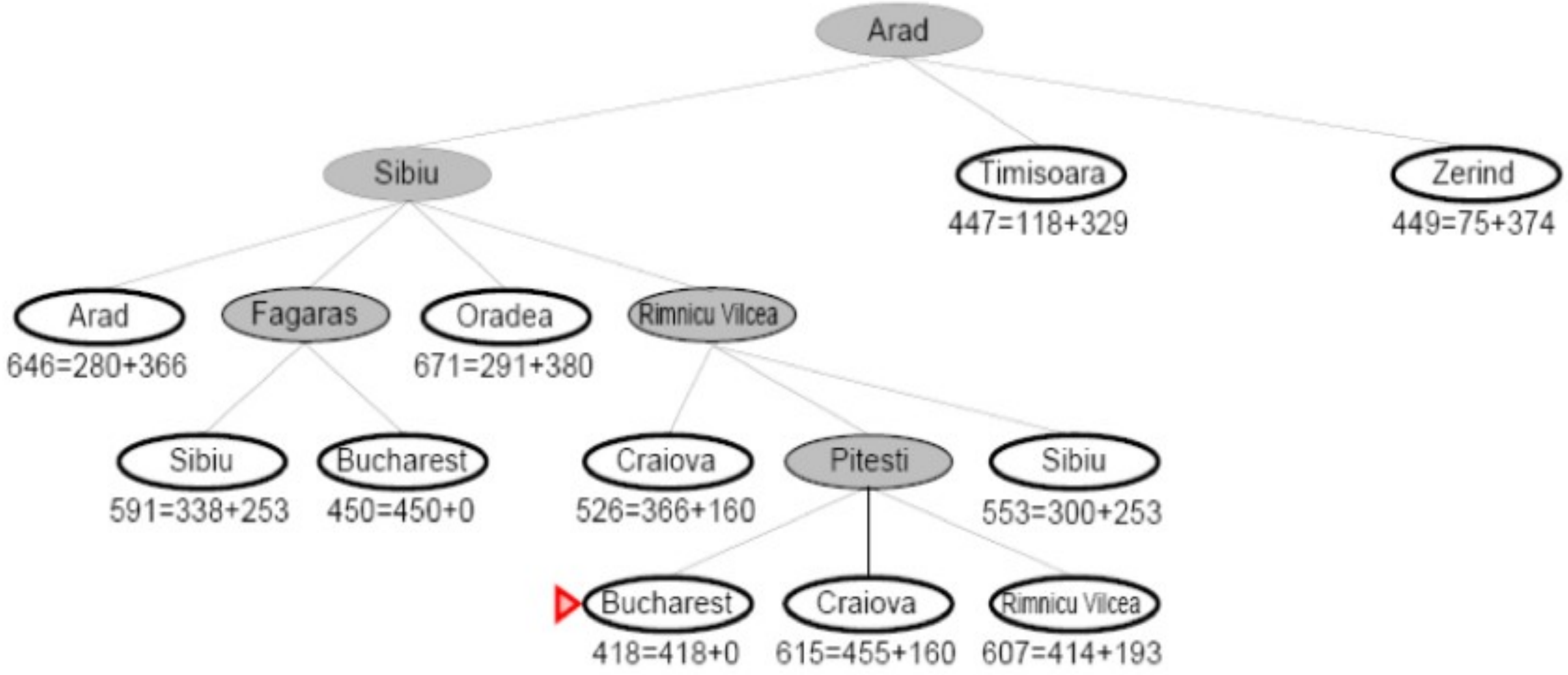
✔ In all nodes expanded Fagaras has lowest f(n) value, so this node is expanded next

Example

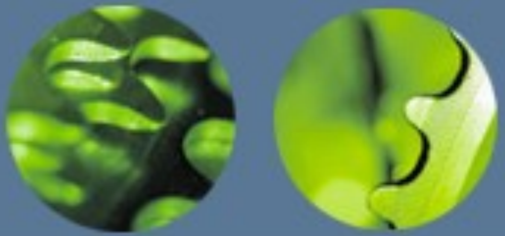


✔ Expanding Fagaras node reaches our destination Bucharest with f(n) value 450. So all the nodes with f(n) values expensive than 450 are avoided as evaluating them would result in cost of path expensive than our path already achieved. The nodes with f(n) value lower than 450 are evaluated next.

Example



✔ Evaluating next node with Petesti with cost 417 (lower than 450 Arad-Sibiu-Fagaras-Bucharest) we reach Bucharest with cost 418 which is lower than our earlier search result. All other nodes are expensive than our new result i.e. cost of 418. Hence our search is stopped here.



Conditions for Optimality: Admissibility and Consistency

- ✓ The first condition we require for optimality is that h(n) be an admissible heuristic.
- ✓ An admissible heuristic is one that never overestimates the cost to reach the goal.
- ✓ Because g(n) is the actual cost to reach n along the current path, and f(n) = g(n) + h(n), we have as an immediate consequence that f(n) never overestimates the true cost of a solution along the current path through n.

Conditions for Optimality: Admissibility and Consistency

- ✓ Notice in particular that Bucharest first appears on the frontier but it is not selected for expansion because its f-cost (450) is higher than that of Pitesti (417).
- ✓ Another way to say this is that there might be a solution through Pitesti whose cost is as low as 417, so the algorithm will not settle for a solution that costs 450.

Conditions for Optimality: Admissibility and Consistency

- ✓ A second, slightly stronger condition called consistency is required only for applications of A* to graph search.
- ✓ A heuristic $h(n)$ is consistent if, for every node n and every successor n' of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n':

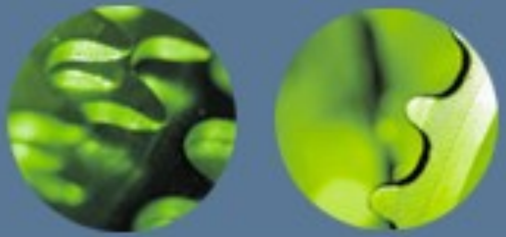
$$h(n) \leq c(n, a, n') + h(n') .$$

Conditions for Optimality: Admissibility and Consistency

- ✓ Every consistent heuristic is also admissible.
- ✓ Consistency is therefore a stricter requirement than admissibility.

Local Search Algorithms and Optimization Problems

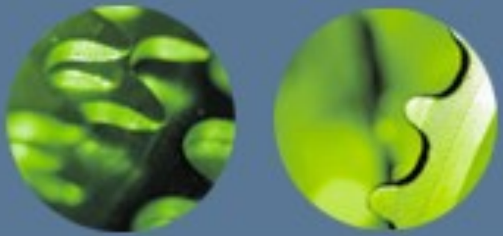
- ✓ If the path to the goal does not matter we might consider a different class of algorithms ones that do not worry about paths at all.
- ✓ Local Search algorithms operate using a single current node rather than multiple paths and generally move only to neighbors of that node.
- ✓ Typically the paths followed by the search are not retained.



Local Search Algorithms and Optimization Problems

- ✓ Although local search algorithms are not systematic they have two key advantages
  - – they use very little memory,usually a constant amount
  - – and they can often find reasonable solutions in large or infinite state spaces for which systematic algorithms are unsuitable

Local Search Algorithms and Optimization Problems

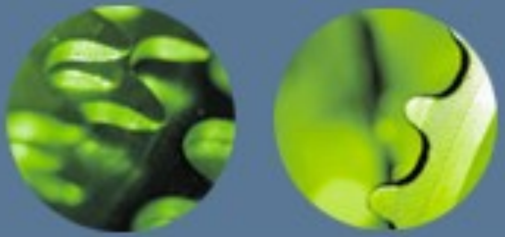


- ✓ In addition to finding goals local search algorithms are useful for solving pure optimization problem in which the aim is to find the best state according to an objective function.
- ✓ For example nature provides an objective function – reproductive fitness – that Darwinian evolution could be seen as attempting to optimize but there is no “goal test” and no “path cost” for this problem.

Local Search Algorithms and Optimization Problems

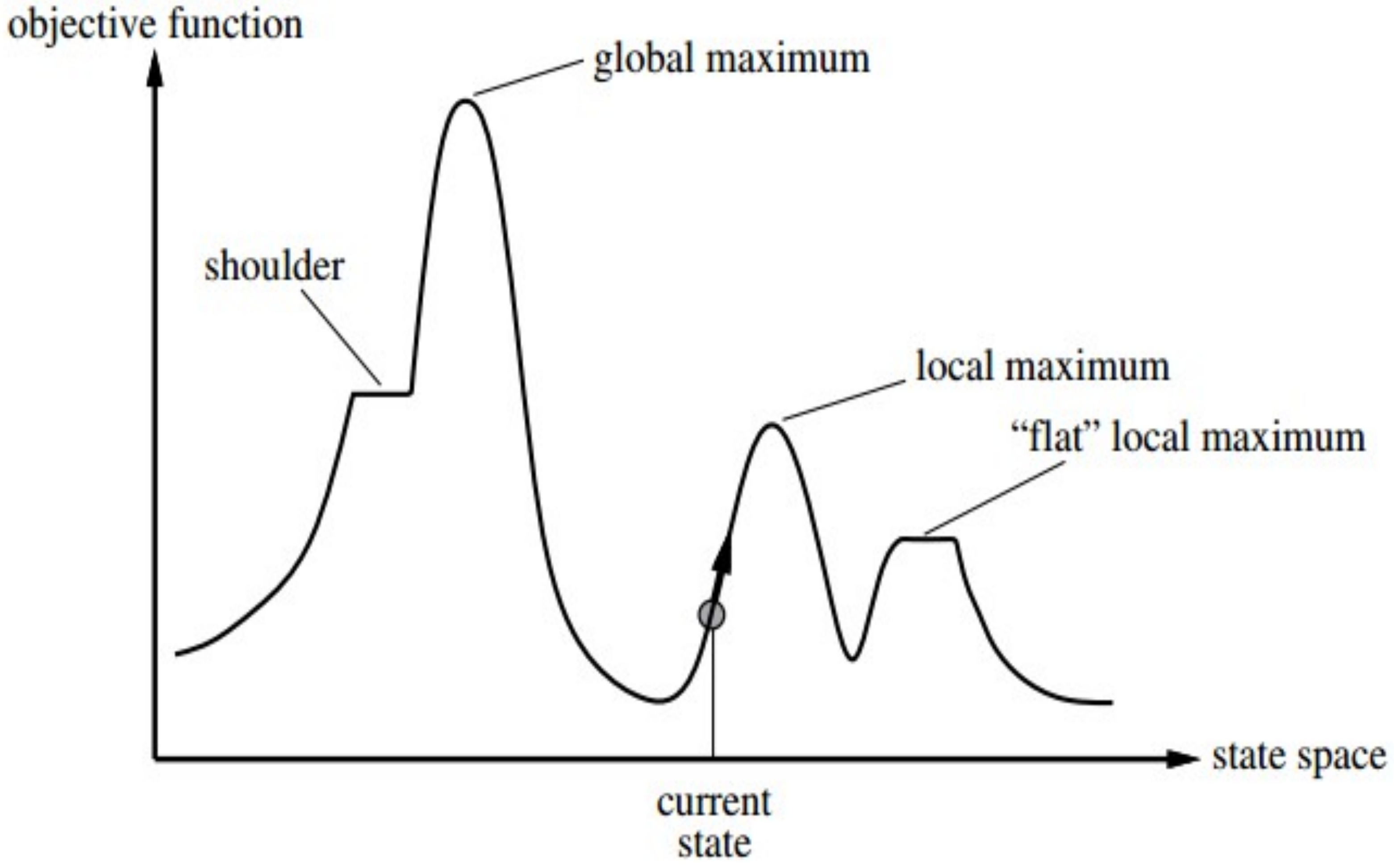
- ✓ Landscape has both ”location” (defined by the state) and “elevation” (defined by the value of the heuristic cost function or objective function).
- ✓ If elevation corresponds to cost then the aim is to find the lowest valley – a global minimum.
- ✓ If elevation corresponds to an objective function then the aim is to find the highest peak – a global maximum.

Local Search Algorithms and Optimization Problems



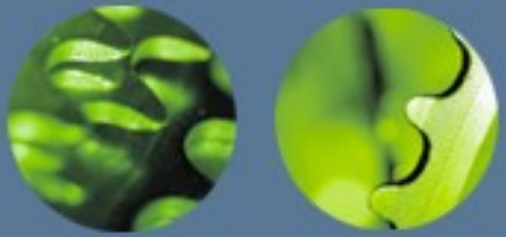
- ✓ Local search algorithm explore this landscape.
- ✓ A local search algorithm always finds a goal if one exists and an optimization algorithm always finds a global minimum/maximum.

Local Search Algorithms and Optimization Problems



Hill Climbing Search

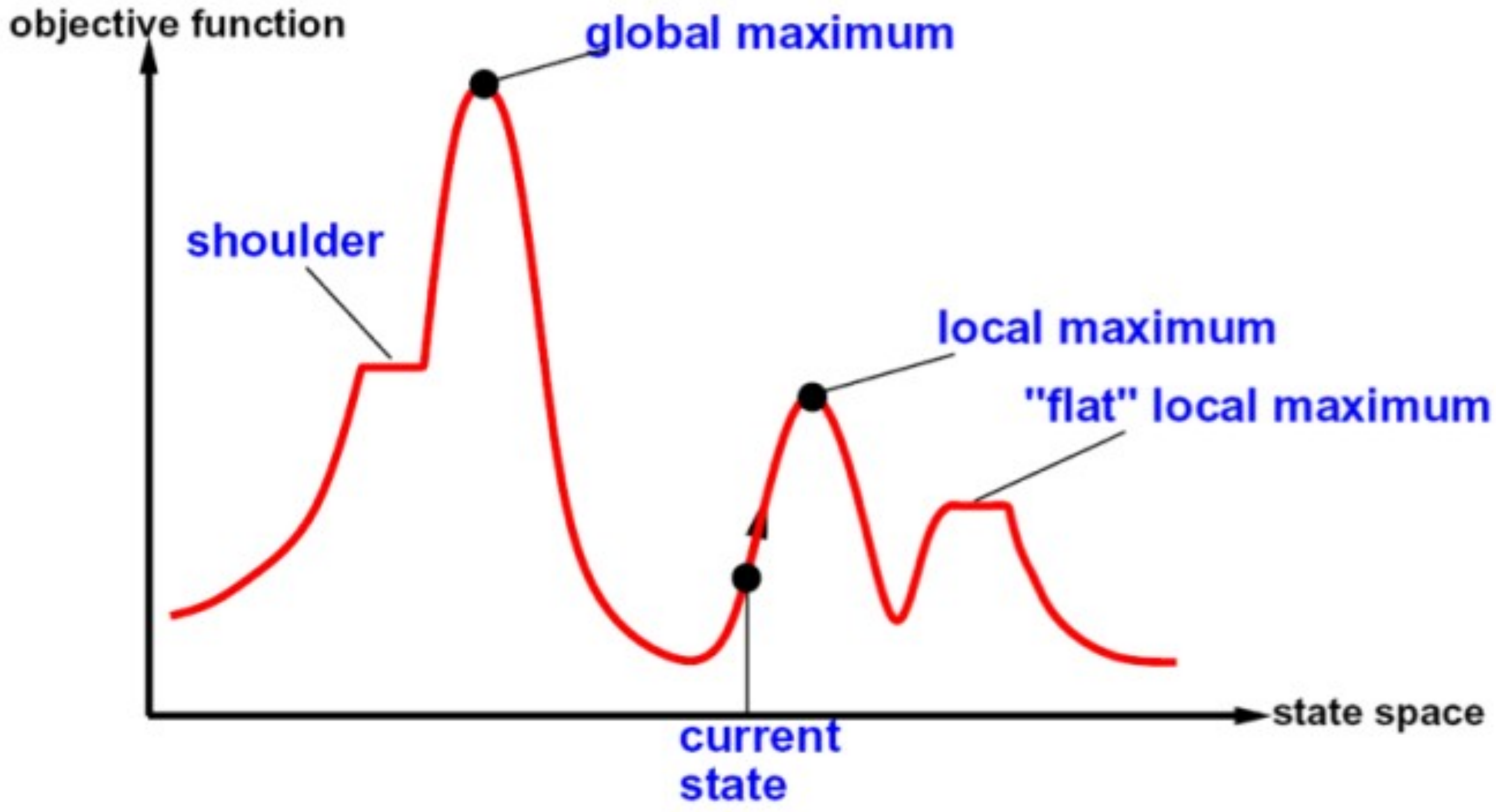
- ✓ Hill climbing search is sometimes compared to climbing Mount Everest in thick fog with amnesia i.e. destined to reach top of Everest but due to thick fog (poor visibility) and forgetting what top of Everest is like, peak lower than top of top of Everest might be considered as top of Everest.



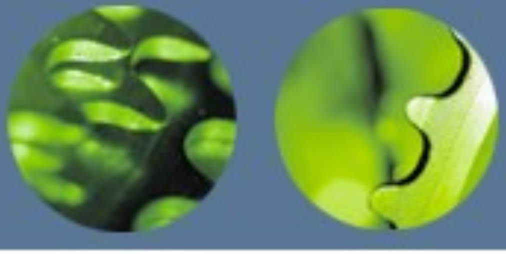
Hill Climbing Search

- ✓ Hill Climbing Search initiates a loop that continuously moves in the direction of increasing value.
- ✓ Terminates when it reaches a “Peak”.
- ✓ Doesn’t maintain a search tree so the current node data structure needs only record the state and its objective function value.
- ✓ Hill climbing does not look ahead beyond the immediate neighbors of the current state.
- ✓ Hill climbing is also called greedy local search sometimes because it grabs a good neighbor state without thinking ahead about where to go next.
- ✓ One move is selected and all other nodes are rejected and are never considered.
- ✓ Halts if there is no successor.

Hill Climbing Search

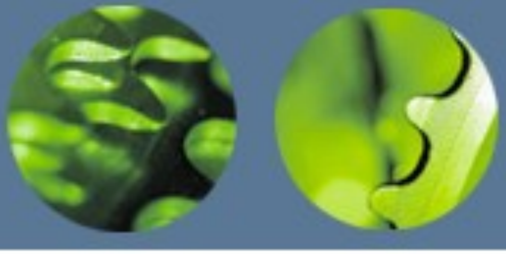


Random restarts – overcomes local maximum
Random sideways moves – escapes shoulders, loop when flat



Simulated Annealing

- ✓ Hill climbing algorithm that never makes downhill moves toward states with lower value or higher cost is guaranteed to be incomplete because it can get stuck on a local maximum.
- ✓ In contrast a purely random walk that is moving to a successor chosen uniformly at random from the set of successors is complete but extremely inefficient.
- ✓ Therefore it seems reasonable to try to combine hill climbing with a random walk in some way that yields both efficiency and completeness.
- ✓ Simulated Annealing is such an algorithm.



Simulated Annealing

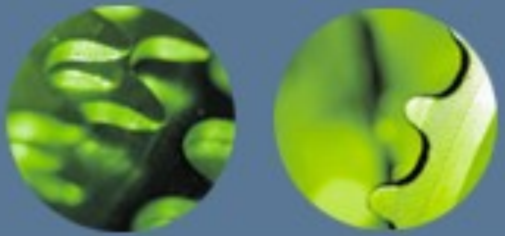
✓ Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency.

✓ Instead of restarting from a random point, we allow the search to take some downhill steps to try to escape local maxima.

✓ A random pick is made for the move

- – If it improves the situation, it is accepted straight away
- – If it worsens the situation, it is accepted with some probability less than 1 which decreases exponentially with the badness of the move i.e. for bad moves the probability is low and for comparatively less bad moves, it is higher.

✓ Probability of downward steps is controlled by temperature parameter.



Simulated Annealing

- ✓ High temperature implies high chance of trying locally "bad" moves, allowing nondeterministic exploration.
- ✓ Low temperature makes search more deterministic (like hill-climbing).
- ✓ Temperature begins high and gradually decreases according to a predetermined annealing schedule.
- ✓ Initially we are willing to try out lots of possible paths, but over time we gradually settle in on the most promising path.
- ✓ If temperature is lowered slowly enough, an optimal solution will be found.
- ✓ In practice, this schedule is often too slow and we have to accept sub-optimal solutions.